A New Leader Election Algorithm based on the WBS Algorithm Dedicated to Smart-cities
Nabil Kadjouh, Ahcène Bounceur, Abdelkamel Tari, Loïc Lagadec, Reinhardt Euler, Bezoui Madani

To cite this version:
Nabil Kadjouh, Ahcène Bounceur, Abdelkamel Tari, Loïc Lagadec, Reinhardt Euler, et al.. A New Leader Election Algorithm based on the WBS Algorithm Dedicated to Smart-cities. 3rd International Conference on Future Networks and Distributed Systems (ICFNDSD ’19), Jul 2019, Paris, France. pp.1-5, 10.1145/3341325.3342014. hal-02303247v2

HAL Id: hal-02303247
https://hal-ensta-bretagne.archives-ouvertes.fr/hal-02303247v2
Submitted on 14 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A New Leader Election Algorithm based on the WBS Algorithm Dedicated to Smart-cities

Nabil Kadjouh  
LIMED laboratory  
University of Bejaia  
Bejaia, Algeria  
nabil.kadjouh@gmail.com

Ahcène Bounceur  
Lab-STICC CNRS UMR 6285  
Université de Bretagne Occidentale  
Brest, France  
Ahcene.Bounceur@univ-brest.fr

Abdelkamel Tari  
LIMED laboratory  
University of Bejaia  
Bejaia, Algeria  
tarikamel59@gmail.com

Loïc Lagadec  
Lab-STICC CNRS UMR 6285  
ENSTA Bretagne  
Brest, France  
loic.lagadec@ensta-bretagne.fr

Reinhardt Euler  
Lab-STICC CNRS UMR 6285  
Université de Bretagne Occidentale  
Brest, France  
Reinhardt.Euler@univ-brest.fr

Madani Bezoui  
LaRoMad Laboratory  
USTHB  
Boumerdes, Algeria  
madani.bezoui@gmail.com

ABSTRACT

One of the interesting techniques for leader election is used in the WBS (Wait Before Starting) algorithm, in which each node in the network will wait for a time corresponding to its value before starting to send the first message to neighbours. This means that the node with the smallest value becomes the leader and it also starts first. This approach is impracticable in the case of real values (case of GPS-coordinates). Also, if the values are very large, the waiting time becomes too long. In this paper, we propose a fast, fault-tolerant and low energy leader election algorithm dedicated to smart-cities, which is based on the technique of waiting before starting, with minimum complexity and in which every node sends one and only one message. Here, the leader is the node with the smallest x-coordinate and the total of sent and received messages is used to represent the global consumption in the network. We give a detailed description of the algorithm, prove its accuracy, discuss its complexity in terms of exchanged messages and evaluate its performance using the CupCarbon simulator. We show that our algorithm is well balanced in terms of energy consumption, it is efficient and adapts well to the increase of the nodes number in the network.

KEYWORDS

Leader election, Smart Cities, Distributed algorithm, Simulation , CupCarbon Simulator.

1 INTRODUCTION AND RELATED WORK

A smart city is a city that collects and uses the data generated by its inhabitants and its infrastructure to improve the quality of life and to optimize resources. Connected meters can provide real-time advice and alerts on energy consumption. The leader in distributed systems is a node with a particular characteristic [3]. In the case of routing, it can be the highest level or the lowest level of energy. In case of the D-LPCN [8, 12], it is the node that starts the algorithm, which is the node having the minimum x-coordinate. Houses can also be equipped with sensors and applications used to adapt the temperature and the brightness automatically depending on the outside weather. On the other hand, houses are connected by an electricity Smart grid that optimizes the production and distribution of electricity to meet customer needs. It also allows control of the pipeline network in real time. And car drivers can park quickly with smart applications that indicate the closest free places in a parking lot. In such networks, the choice of a coordinating node is indispensable in order to ensure synchronization between the network nodes. We call this coordinator the leader which is a node with special properties. In the case of border detection, it can be the node with the largest or the smallest x or y coordinates. In the rooting case, it can be the node having a high level of battery, so everything depends on the application. The challenge here is to find a fast and least expensive way to elect the leader.

In the literature, few techniques have been used to elect the leader according to the different types of a network. In the network over a complete graph, in which all nodes can communicate directly, Bully’s algorithm is the most known [13] since it is very easy to apply. In this algorithm, a node P starts the election by sending an Election message to all nodes having a higher identifier then P. If no one responds after a specific period, P becomes the leader, otherwise if one of the higher-ups answers, it takes over. The main drawback of this algorithm is that it works only with complete graphs. In addition, every time a node recovers from a crash failure, it initiates an election, which consumes system resources. In the same context, many improvements have been based on Bully’s algorithm in order to have a better solution. One of the most used solutions in this type of network is the LMF algorithm (Local minimum finding) [11], in which each node assumes that its value is the minimum and broadcasts it, then waits. If it receives...
a value lower its own, it looses its chance to become the leader, otherwise, it becomes the leader. The idea of this algorithm is useful if the use conditions are verified. In another type of networks all nodes are linked by a single path, and the best-known example is the ring algorithm and its improvements [10]. The main idea is that nodes are ordered so that every node knows its successor. When a node P detects the crash of the leader, it sends its value in the stack to its successor node. If the successor receives the message, it adds its value to the stack and an election message will circulate over the ring to arrive at the election initiator. The election time is the same in the best and worst case because this idea requires the traversal of all nodes, and therefore electing a new leader takes a longer time.

In the random topology, the idea is to find an optimal path linking all nodes which are used to elect the leader, and the known structure representing this path is the spanning tree. In this context, many strategies are based on the construction of a spanning tree to elect the leader. LOGO [7], BrOGO [4] and DoTRo [6] are three algorithms which recently appeared to elect the leader in a random graph.

The basic idea of the BrOGO algorithm (Branch Optima to Global Optimum) is the construction of a spanning tree used to route the best value from the leaves to the root.

The idea of the LOGO algorithm (Local Optima to Global Optima) is to make two types of election, first local then global. The local election is based on the LMF algorithm as previously seen, in order to determine the local minima. Then in the global election, the information of local leaders will be routed by construction of the spanning tree from the local leaders to a reference node which is the root of the spanning tree, to select the global leader. This technique is fault-tolerant but it is relatively slow.

Another use of the LMF algorithm is in the DoTRo (Domination Tree Routing) algorithm to determine the local leaders, each one of which becomes the root of a spanning tree. If two trees meet each other then the tree routing the minimum value will continue its process, while the other tree will stop it. Finally, the leader will be the root of the lonely spanning tree.

In this context, we assume that the network is composed of non-mobile nodes [1], we propose a new leader election algorithm based on the waiting time function dedicated to smart cities. Its idea is extracted from an algorithm recently published and called WBS (Wait Before Start) [5] which uses the waiting time to elect the leader. The principle is interesting and easy to implement but only in very special cases. We dedicate Section 2 to describe the WBS algorithm. In Section 3 we show the detailed description of the proposed algorithm. Section 4 contains the representation of SenScript and the CupCarbon simulation environment and in Section 5, we present the results and a discussion of the simulation. Finally, Section 6 concludes the paper.

2 THE WBS ALGORITHM

In this algorithm, the authors assume that the leader is the node having the minimum value among all nodes in the network. In order to elect this node, each node must wait for a duration which is proportional to its value before sending its message, which means that the leader is the node who starts first to send its unique message to inform the other nodes using the flooding process for example. The example in Figure 1 represents an illustration of this algorithm, where we assume that there are 3 nodes S1, S2 and S3 with values 8, 2 and 5 respectively. Then S1, S2 and S3 will wait 8, 2 and 5 seconds respectively. The node S2 after 2 seconds will send a message first and becomes the leader.

![Figure 1: An illustration of the WBS algorithm.](image)

The pseudo-code of this approach is given by Algorithm 1. The WBS algorithm needs as input data: the value of the x-coordinate and gt, and the boolean Leader as output data. At the beginning each node is initialized as a leader and will wait for the duration of x * gt (Algorithm 1 lines 1:3); then if it does not receive any message when waiting, it becomes the leader and broadcasts its message to stops (lines 4:10). But if it receives a message it roots it and stops (lines 11:16). Figure 1 shows that the node with the smallest x-coordinate becomes the leader and constructs a spanning tree to inform the other nodes that it is the leader.

**Algorithm 1 WBS:** The pseudo-code of the WBS algorithm.

| Input:  | x, gt |
| Output: | leader |

1. leader = true
2. once = false
3. \( t = x \ast gt \)
4. while (true) do
5. \( rx = \text{read}(t) \)
6. if \( (rx == \text{null}) \) then
7. leader = true
8. send(A, *)
9. stop()
10. end if
11. if \( ((rx == A) \) and \( (once == \text{false}) \) then
12. once = true
13. send(A, *)
14. stop()
15. end if
16. end while

The idea of this algorithm is simple and easy to implement in cases where the values are integers, but we cannot apply it if the values are negative, real, or if the numbers are very large.

3 THE PROPOSED ALGORITHMS

In this section, we give a detail description of our algorithm. First, we explain the global idea of the protocol, then before we describe the different steps that compose it, where all used variables and methods are presented in Table 1 and Table 2. Finally, we give an illustration example. This algorithm is dedicated to the city, so we have to know the x-coordinates of any marker in the city, for example, a round-point, tricolour lights, billboard, bus stop or any static object in the city. Here, we consider as a leader the node having the smallest x-coordinate. A waiting time is imposed on all
nodes according to their x-coordinates so that the first node who sends its message becomes the leader. In another way, the leader here is the node that waits for the least duration and that sends its message first. The algorithm needs as input data: the value of $x$-coordinate and $x$.marker, also the boolean leader and id.leader as output data (cf. Algorithm 2). This protocol can be summarized in 5 main steps:

- **Step 1:** After having obtained its identifier, each node initializes the boolean values: leader and once to False, to say that it is not a leader and it did not send or receive any message until this moment (lines 1:3).

- **Step 2:** Calculating the new $x$-coordinate according to the static marker, which is the distance between the node and the marker on the x-axis (line 4).

- **Step 3:** Calculating wt, the waiting time according to the new $x$-coordinate as follows: first eliminate the comma in the x-coordinate by multiplying its value by 10000000 (line 5). Then use the logarithmic function to make this value very small in order to minimize the waiting time (line 6). Finally extract the waiting time which is the integer part of the result (line 7).

- **Step 4:** After waiting for wt milliseconds (line 8), if there is no receipt message, the node becomes the leader and informs the others by sending its identifier (lines 9:13). Otherwise, it reads the identifiers of the transmitter and of the leader (lines 14:17).

- **Step 5:** If the node did not receive any message it will route the leader and its identifiers to the others except the transmitter, and stops (lines 18:24).

For the demonstration of the algorithm’s progress, Figure 2 shows a representation of a random network topology with 100 nodes, on a map of a real city part in which we have fixed a point represented by a cross symbol. After calculating the new x-coordinates and the waiting time of each node according to its position, the first node that executes its program becomes the leader (cf. Figure 2 (c)) and informs its neighbours by sending a message to them whenever a node receives a message from its neighbour, it will send it to its neighbours and stops as a non leader node (cf. Figure 2 (d)). Finally the spanning tree of Figure 2 (e) represents the path of information from the leader to all the nodes of the network.

### Table 1: Description of the variables.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>x-coordinate of the node</td>
</tr>
<tr>
<td>$x$.marker</td>
<td>x-coordinate of the marker in the city</td>
</tr>
<tr>
<td>leader</td>
<td>A boolean variable. It is True if a node is the leader and False otherwise</td>
</tr>
<tr>
<td>id.leader</td>
<td>Leader identifier</td>
</tr>
<tr>
<td>id</td>
<td>Node identifier</td>
</tr>
<tr>
<td>$x$.new</td>
<td>The new x-coordinate according to the marker</td>
</tr>
<tr>
<td>once</td>
<td>A boolean variable. It is True if a node send or receive its first message and False otherwise</td>
</tr>
<tr>
<td>tmp</td>
<td>Temporary variable</td>
</tr>
<tr>
<td>wt</td>
<td>The waiting time</td>
</tr>
<tr>
<td>transmitter</td>
<td>Id of the transmitter node</td>
</tr>
</tbody>
</table>

### Table 2: Description of the methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getld()</td>
<td>Returns the node identifier</td>
</tr>
<tr>
<td>log(x)</td>
<td>Returns the neperian logarithm of x</td>
</tr>
<tr>
<td>int(x)</td>
<td>Returns the integer part of x after rounding down</td>
</tr>
<tr>
<td>stop()</td>
<td>Stops the program execution</td>
</tr>
<tr>
<td>send( msg, b)</td>
<td>Sends msg to the sensor node having b identifier , or in a broadcast (if b = ‘)</td>
</tr>
<tr>
<td>send( msg, b)</td>
<td>Sends msg to all sensors except node having b identifier</td>
</tr>
<tr>
<td>read()</td>
<td>Waiting for receipt of message.</td>
</tr>
<tr>
<td>read(wt)</td>
<td>Waiting for receipt of message. If there is no received message after wt milliseconds then the execution will continue and go to the next instruction</td>
</tr>
</tbody>
</table>

### Algorithm 2 The pseudo-code of the proposed algorithm.

**Input:** $x$, $x$.marker;  
**Output:** leader, id.leader;  

```plaintext
1. id = getld();
2. leader = false;
3. once = false;
4. $x$.new = $x$.x.marker;
5. $tmp$ = $x$.new * 10000000;
6. $tmp$ = log ($wt$);
7. $wt$ = int ($tmp$);
8. msg = read ($wt$);
9. if (msg == null) then
10. id.leader = id;
11. leader = true;
12. send (id.leader + ‘|’ + id, ‘); 
13. once = true;
14. else
15. transmitter = read();
16. id.leader = read();
17. end if
18. while (true) do
19. if (once == false) then
20. send (id.leader + ‘|’ + id, ‘, transmitter); 
21. once = true;
22. stop();
23. end if
24. end while
```

### 4 THE SIMULATION ENVIRONMENT

In our work, we have used the CupCarbon simulator [2] [9], since it is dedicated to Smart City and Internet of Things Wireless Sensor Networks (SCI-WSN) and also open source, including the use the OpenStreetMap (OSM) in which we can drop our virtual network on a real city in 2 or 3D. Also, it offers an easy visual interface that allows to implement distributed algorithms and visualize all simulation events during the execution of the simulation process. In the context of leader election, it is possible to highlight sent/received messages with different colours for each node and at any point in the simulation. The result (the leader’s node) can be a marked node.
number of messages received by our algorithm is the smallest in all cases, and it represents the sum of the number of neighbors of all the nodes. The results of the BrOGO algorithm are close to that of our algorithm compared to the results of LOGO and DoTRo. The results of the simulation show that the energy consumption is balanced in our algorithm, independent of the network size.

6 CONCLUSION

In this paper, we have presented a new algorithm for leader election dedicated to Smart cities based on the WBS technique which works only with integer values. We have added a waiting time adjustment mechanism that allows us to use any type of value. We have simulated our algorithm on the CupCarbon environment. The results show that our algorithm is fast and less expensive with a complexity equal to one and only one message sent by each node and an average number of neighbors for the messages received by each node. The results of the simulation show that the energy consumption is balanced in our algorithm and independent of the network size.

REFERENCES


